СПбГУТ им. проф. М.А. Бонч–Бруевича
Кафедра программной инженерии и вычислительной техники (ПИ и ВТ)

ПРОГРАММИРОВАНИЕ

Единственный способ изучать новый язык программирования – писать на нем программы.

Брайэн Керниган

Лекция 9-10: *Указатели и массивы*

1. Указатели и массивы
2. Массив указателей
3. Строки
4. Примеры обработки массивов

Язык Си как острая бритва: с его помощью можно сделать изящное произведение искусства или кровавое месиво.

Брайэн Керниган

Санкт–Петербург,

1. Указатели и массивы

- В языке **Си** **массивы** и **указатели** тесно связаны. С помощью **указателей** мы также легко можем манипулировать **элементами массива**, как и с помощью **индексов**.
- **Имя массива** без индексов в **Си** является **адресом его первого элемента**. Соответственно через операцию разыменования мы можем получить значение по этому адресу:

```
int a[] = {1, 2, 3, 4, 5};
printf("a[0] = %d", *a);    // a[0] = 1
```

- Мы можем пробежаться по всем элементом массива, прибавляя к адресу определенное

```
#include <stdio.h>
int main(void)
{
    int a[5] = {1, 2, 3, 4, 5};
    for(int i=0;i<5;i++)
    {
        printf("a[%d]: address=%p \t value=%d \n", i, a+i, *(a+i));
    }
    return 0;
}
```

```
a[0]: address=0060FE98    value=1
a[1]: address=0060FE9C    value=2
a[2]: address=0060FEA0    value=3
a[3]: address=0060FEA4    value=4
a[4]: address=0060FEA8    value=5
```

То есть, например, адрес второго элемента будет представлять выражение **a+1**, а его значение - ***(a+1)**.

- Со сложением и вычитанием здесь действуют те же правила, что и в операциях с указателями. Добавление единицы означает прибавление к адресу значения, которое равно размеру типа массива. Так, в данном случае массив представляет тип int, размер которого, как правило, составляет 4 байта, поэтому прибавление единицы к адресу означает увеличение адреса на 4. Прибавляя к адресу 2, мы увеличиваем значение адреса на 4 * 2 =8. И так далее.

- В то же время **имя массива** это не стандартный указатель, мы не можем изменить его адрес, например, так:

```
int a[5] = {1, 2, 3, 4, 5};
a++;          // так сделать нельзя!!!
int b = 8;
a = &b;       // так тоже сделать нельзя
```

- Имя массива всегда хранит адрес самого первого элемента. И нередко для перемещения по элементам массива используются отдельные указатели:

```
int a[5] = {1, 2, 3, 4, 5};
int *ptr = a;
int a2 = *(ptr+2);
printf("value: %d \n", a2); // 3
```

Здесь указатель **ptr** изначально указывает на первый элемент массива. Увеличив указатель на **2**, мы пропустим **2** элемента в массиве и перейдем к элементу **a[2]**.

- С помощью указателей легко перебрать массив:

```
int a[5] = {1, 2, 3, 4, 5};
for(int *ptr=a; ptr<=&a[4]; ptr++)
{
    printf("address=%p \t value=%d \n", ptr, *ptr);
}
```

Так как указатель хранит адрес, то мы можем продолжать цикл, пока адрес в указателе не станет равным адресу последнего элемента.

Результатом использования **указателей для массивов** является **меньшее количество используемой памяти** и **высокая производительность!!!**

Указатели и массивь

Пусть есть массив:

int A[5] = {1, 2, 3, 4, 5};

Мы уже показали, что указатели очень похожи на массивы. В частности, массив хранит адрес, откуда начинаются его элементы. Используя указатель можно также получить доступ до элементов массива:

int *p = A;

Тогда вызов **A[3]** эквивалентен вызову ***(p + 3)**.

На самом деле **оператор [ ]** является «*синтаксическим сахаром*» – он выполняет точно такую же работу.

То есть вызов **A[3]** также эквивалентен вызову ***(A + 3)**:

```
#include <conio.h>
#include <stdio.h>
void main()
{
   int A[5] = {1, 2, 3, 4, 5};
   int *p = A;
   printf("%d\n", A[3]);
   printf("%d\n", *(A + 3));
   printf("%d\n", *(p + 3));
   getch();
}
```

```
4
4
4
```

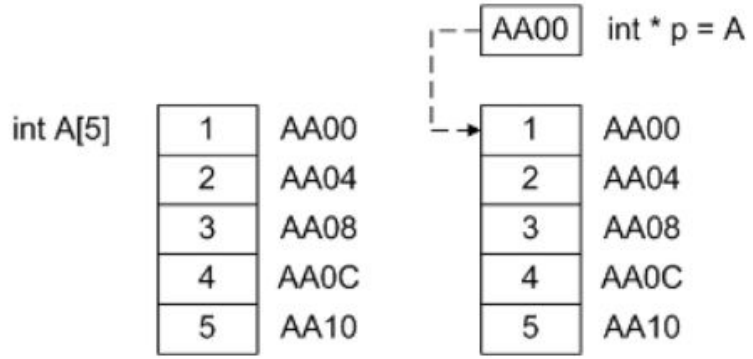
```
#include <stdio.h>
void main()
{
   int A[5] = {1, 2, 3, 4, 5};
   int *p = A;
   printf("%d\n", *(A+1));
   printf("%d\n", *(p+1));
   getch();
}
```

Тем не менее, важно понимать: **указатели – это не массивы!**

Указатель – это переменная, поэтому можно написать pa=a или pa++. Но имя массива – не является переменной, и записи вроде a=pa или a++ не допускаются.

Массив – непосредственно указывает на первый элемент, **указатель-переменная**, которая хранит адрес первого элемента.

Тогда почему возможна следующая ситуация (см. код в



NB: «***Синтаксический сахар***» *(syntactic sugar) в языке программирования — это синтаксические возможности, применение которых не влияет на поведение программы, но делает использование языка более удобным для человека.*

Код в зелёном прямоугольнике – правильный. Он будет работать. Дело в том, что ***компилятор подменяет массив на указатель***.

Данный пример работает, потому что мы действительно работаем с **указателем** (хотя помним, что массив отличается от указателя).

То же самое происходит и при **вызове функции**.

Если **функция** требует указатель, то можно передавать в качестве аргумента массив, так как он будет подменён указателем.

В **Си** существует одна занимательная особенность.

Если **A[i]** это всего лишь «*синтаксический сахар*», и **A[i] == *(A + i)**, то от смены слагаемых местами ничего не должно поменяться, т. е. **A[i] == *(A + i) == *(i + A) == i[A]**

Как бы странно это ни звучало, но это действительно так.

Следующий код вполне валиден:

```
int a[] = {1, 2, 3, 4, 5};
printf("%d\n", a[3]);
printf("%d\n", 3[a]);
```

Указатели и массивы

Различия между указателями и массивами

1. **Основное различие** возникает при использовании оператора **sizeof**.

При использовании в *фиксированном массиве*, оператор **sizeof** возвращает размер всего массива:

*длина_массива * размер_элемента*

При использовании с указателем, оператор **sizeof** возвращает размер адреса памяти (в **байтах**).

Например:

```
#include <stdio.h>
int main()
{
   int array[4] = { 5, 8, 6, 4 };
   // выведется sizeof(int) * длина array:
   printf("%d\n", sizeof(array));
   int *ptr = array;
   printf("%d\n", sizeof(ptr));    // выведется размер указателя
   return 0;
}
```

Вывод: *Фиксированный массив знает свою длину, а указатель на массив — нет*.

2. **Второе различие** возникает при использовании оператора адреса &.

Используя адрес указателя, мы получаем адрес памяти переменной-указателя. Используя адрес массива, возвращается указатель на целый массив. Этот указатель также указывает на первый элемент массива, но информация о типе отличается.

Рассмотрим **инициализацию указателей** типа **char**:

```
char  *ptr = "hello, world";
```

- Переменная ***ptr** является указателем, а не массивом.
- Поэтому строковая константа **"hello, world"** не может храниться в указателе ***ptr**.
- Тогда возникает вопрос, **где хранится строковая константа?**
- Для этого следует знать, **что происходит**, когда компилятор встречает **строковую константу**:
  - Компилятор создает так называемую **таблицу строк**.
  - В ней он **сохраняет строковые константы**, которые встречаются ему по ходу чтения текста программы.
- Следовательно, когда встречается объявление с инициализацией, то компилятор сохраняет **"hello, world"** в таблице строк, а указатель ***ptr** записывает ее адрес.

Поскольку **указатели** сами по себе являются **переменными**, их можно хранить в массивах, как и переменные других типов. Получается ***массив указателей***.

2. Массив указателей

Массив указателей фиксированных размеров вводится одним из следующих определений:

тип *имя_массива [размер];

тип *имя_массива [ ] = {инициализатор};

тип *имя_массива [размер] = {инициализатор};

- В данной инструкции **тип** может быть как одним из базовых типов, так и **производным типом**;
- **имя_массива** – идентификатор, определяемый пользователем по правилам языка **Си** ;
- **размер** – **константное выражение**, вычисляемое в процессе трансляции программы;
- **инициализатор** – **список** в фигурных скобках **значений элементов заданного типа** (т.е. тип).

Рассмотрим **примеры**:

```
int  data[7]; // обычный массив
int  *pd[7];  // массив указателей
int  *pi[ ] = { &data[0], &data[4], &data[2] };
```

- В приведенных примерах каждый элемент массивов **pd** и **pi** является указателем на объекты типа **int**.
- Значением каждого элемента **pd[j]** и **pi[k]** может быть адрес объекта типа **int**.
- Все элементы массива **pd** указателей не инициализированы.
- В массиве **pi** три элемента, и они инициализированы адресами конкретных элементов массива **data**.
- В **случае обработки строк текста** *они, как правило, имеют различную длину, и их нельзя сравнить или переместить одной элементарной операцией в отличие от целых чисел*.
- В этом случае эффективным средством является **массив указателей**.
- Например:
  - ✔ если сортируемые строки располагаются в одном длинном символьном массиве вплотную — ***начало одной к концу другой***, то **к каждой строке можно обращаться по указателю на ее первый символ**.
  - ✔ Сами же **указатели** можно **поместить в массив**, т.е. создать **массив указателей**.
  - ✔ ***Две строки можно сравнить, рассмотрев указатели на них***.
- Массивы указателей часто используются при работе со строками.
- **Пример** ***массива строк о студенте***, задаваемый с помощью массива указателей:

```
char  *ptr[ ] = {
 "Surname",  //Фамилия
 "Name",     // Имя
 "group",    // группа
 "ACOUY"     // специальность
};
```

- С помощью **массива указателей** можно ***инициализировать строки различной длины***.
- Каждый из **указателей массива указателей** указывает на ***одномерный массив символов (строку)*** независимо от других указателей.

Массив указателей.

Массив указателей как указывалось выше (см. пред. слайд) определяется одним из трех способов:

- **тип *имя_массива [размер];**
- **тип *имя_массива [] = {инициализатор};**
- **тип *имя_массива [размер] = {инициализатор};**

Используем все эти способы:

```
int a[] = {1, 2, 3, 4};
int *p1[3];
int *p2[] = { &a[1], &a[2], &a[0] };
int *p3[3] = { &a[3], &a[1], &a[2] };
```

Массив указателей **p1** состоит из трех элементов, но он не инициализирован и является пустым.

Массивы **p2** и **p3** в качестве элементов хранят адреса на элементы массива a.

3. Строки

Ранее мы рассмотрели, что **строка** по сути является **массивом символов**, окончанием которого служит нулевой символ '**\0**'.

И фактически строку можно представить в виде массива:

```
char[] hello = "Hello World";
```

Но в языке **Си** также для представления строк можно использовать указатели на тип char:

```
char *hello = "Hello World!";
printf("%s", hello);
```

Оба **определения строки** – с помощью *массива* и *указателя* будут равнозначны.

Соответственно массив указателей типа char представляет собой набор строк (см. пример на **Си** справа-вверху).

При определении массива символов необходимо сообщить компилятору требуемый размер памяти.

```
char m[82];
```

- В программе **строки** могут **определяться** следующим образом:
  - как строковые константы;
  - как массивы символов;
  - через указатель на символьный тип;
  - как массивы строк.
- Кроме того, должно быть предусмотрено выделение памяти для хранения строки.

❖ *Любая последовательность символов, заключенная **в двойные кавычки** " ", рассматривается как **строковая константа**.*

- Для корректного вывода любая строка должна заканчиваться ***нуль-символом*** '**\0**'(см. пред. лекцию) .
- **Строковые константы** размещаются в статической памяти. Начальный адрес последовательности символов в двойных кавычках трактуется как адрес строки.
- **Строковые константы** часто используются для осуществления диалога с пользователем в таких функциях

```
#include <stdio.h>
int main(void)
{
   char *fruit[] = {"apricot", "apple", "banana", "lemon", "pear", "plum"};
   int n = sizeof(fruit)/sizeof(fruit[0]);
   for(int i=0; i<n; i++)
   {
      printf("%s \n", fruit[i]);
   }
   return 0;
}
```

```
apricot
apple
banana
lemon
pear
plum
```

Компилятор также может самостоятельно определить размер массива символов, если инициализация массива задана при объявлении строковой константой:

```
char m2[]="Горные вершины спят во тьме ночной.";
char m3[]={'Т','и','х','и','е',' ','д','о','л','и','н','ы',' ','п','о','л','н',\
'ы',' ','с','в','е','ж','е','й',' ','м','г','л','о','й','\0'};
 /* Обратный слэш "\" служит для переноса содержимого операторной строки на новую строку (для удобства восприятия) */
```

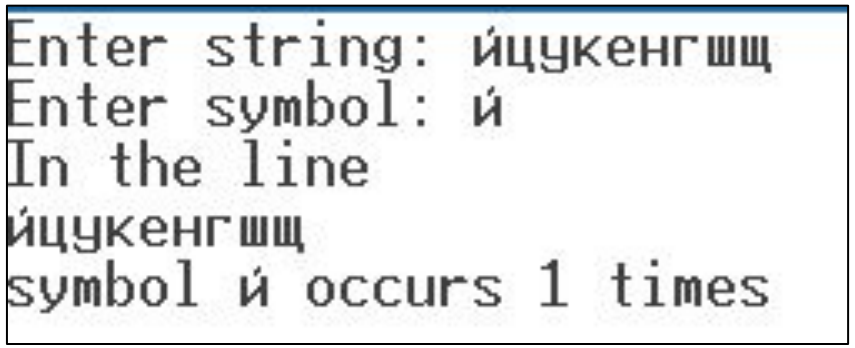
В этом случае имена **m2** и **m3** являются указателями на первые элементы массивов:

m2	эквивалентно	**&m2[0]**
m2[0]	эквивалентно	**‘Г’**
m2[1]	эквивалентно	**‘о’**
m3	эквивалентно	**&m3[0]**
m3[2]	эквивалентно	**‘х’**

Строки

Пример: *посчитать количество введенных символов во введенной строке.*

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s[80], sym;
  int count, i;
  printf("Enter string: ");
  gets(s);                // Функции ввода строк
  printf("Enter symbol: ");
  sym = getchar(); // Функция ввода символов
  count = 0;
  for (i = 0; s[i] != '\0'; i++)
  {
    if (s[i] == sym)
      count++;
  }
  printf("In the line\n");
  puts(s);                // Вывод строки
  printf("symbol ");
  putchar(sym);    // Вывод символа
  printf(" occurs %d times", count);
  getchar();
  return 0;
}
```



Строки

Основные функции стандартной библиотеки **string.h**

Функция	Описание
`char *strcat(char *s1, char *s2)`	присоединяет s2 к s1, возвращает s1
`char *strncat(char *s1, char *s2, int n)`	присоединяет не более n символов s2 к s1, завершает строку символом '\0', возвращает s1
`char *strcpy(char *s1, char *s2)`	копирует строку s2 в строку s1, включая '\0', возвращает s1
`char *strncpy(char *s1, char *s2, int n)`	копирует не более n символов строки s2 в строку s1, возвращает s1;
`int strcmp(char *s1, char *s2)`	сравнивает s1 и s2, возвращает значение 0, если строки эквивалентны
`int strncmp(char *s1, char *s2, int n)`	сравнивает не более n символов строк s1 и s2, возвращает значение 0, если начальные n символов строк эквивалентны
`int strlen(char *s)`	возвращает количество символов в строке s
`char *strset(char *s, char c)`	заполняет строку s символами, код которых равен значению c, возвращает указатель на строку s
`char *strnset(char *s, char c, int n)`	заменяет первые n символов строки s символами, код которых равен c, возвращает указатель на строку s

Пример использования функций:

```
#include <stdio.h>
#include <string.h>
#define _CRT_SECURE_NO_WARNINGS /* для совместимости с классическими функциями */
int main()
{
  char m1[80] = "first line";
  char m2[80] = "second line";
  char m3[80];
  strncpy(m3, m1, 6);  // не добавляет '\0' в конце строки
  puts("Result strncpy(m3, m1, 6)");
  puts(m3);
  strcpy(m3, m1);
  puts("Result strcpy(m3, m1)");
  puts(m3);
  puts("Result strcmp(m3, m1) равен");
  printf("%d", strcmp(m3, m1));
  strncat(m3, m2, 5);
  puts("Result strncat(m3, m2, 5)");
  puts(m3);
  strcat(m3, m2);
  puts("Result strcat(m3, m2)");
  puts(m3);
  puts("The number of characters in line m1 is equal to  strlen(m1) : ");
  printf("%d\n", strlen(m1));
  _strnset(m3, 'f', 7);
  puts("Result strnset(m3, 'f', 7)");
  puts(m3);
  _strset(m3, 'k');
  puts("Result strnset(m3, 'k')");
  puts(m3);
  getchar();
  return 0;
}
```

```
Result strncpy(m3, m1, 6)
first
Result strcpy(m3, m1)
first line
Result strcmp(m3, m1) Ёртхэ
0Result strncat(m3, m2, 5)
first linesecon
Result strcat(m3, m2)
first lineseconsecond line
The number of characters in line m1 is equal to  strlen(m1) :
10
Result strnset(m3, 'f', 7)
fffffffineseconsecond line
Result strnset(m3, 'k')
kkkkkkkkkkkkkkkkkkkkkkkkk
```

4. Указатели на указатели

❖ *В языке программирования **Си** предусматриваются ситуации, когда **указатели указывают на указатели**. Такие ситуации называются **многоуровневой адресацией**.*

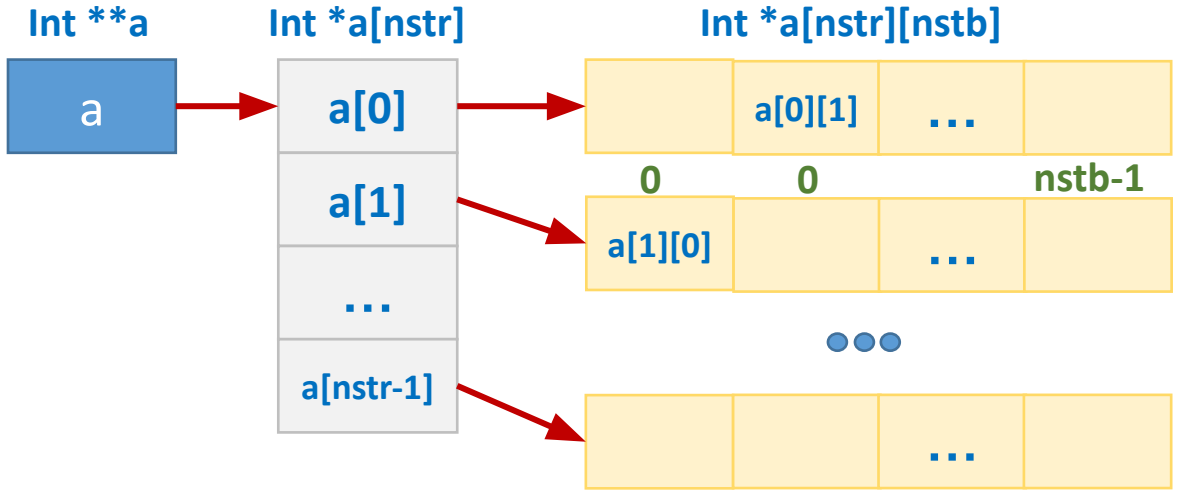
Пример объявления **указателя на указатель**:

```
int  **ptr2;
```

- ❑ В приведенном объявлении **ptr2 – это **указатель на указатель** на число типа int.
- ❑ При этом наличие **двух звездочек свидетельствует о том, что имеется **двухуровневая адресация**.
- ❑ Для получения **значения** конкретного числа следует выполнить следующие действия:

```
int  x = 88, *ptr,  **ptr2;
ptr = &x;
ptr2 = &ptr;
printf("%d", **ptr2);
```

- ▪ В результате в выходной поток (на дисплей пользователя) будет выведено число **88**.
- ▪ В приведенном фрагменте переменная ***ptr** объявлена как указатель на целое число, а ****ptr2** – как указатель на указатель на целое.
- ▪ Значение, выводимое в выходной поток (число **88**), осуществляется операцией разыменования указателя ****ptr2**.
- ❑ Для многомерных массивов указатели указывают на адреса элементов массива построчно (**рассмотрим на последующих занятиях**).



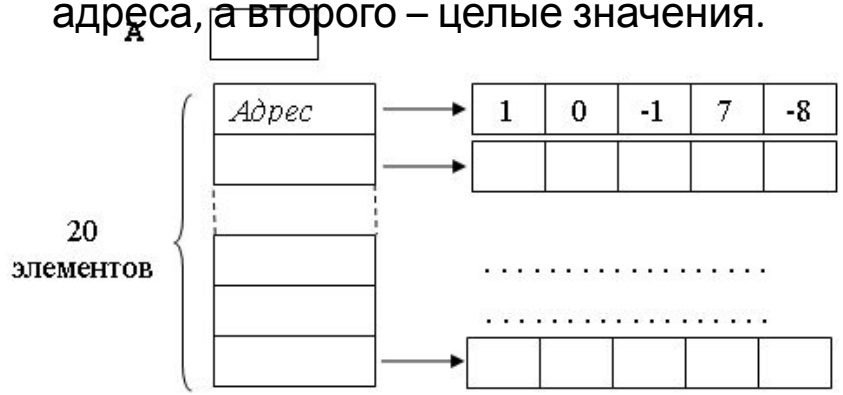
❖ ***Двумерный массив** (**матрица**) – одномерный массив одномерных массивов.*

<тип элементов> <имя массива>[количество][количество];

- ▪ Указывается количество элементов в одномерном массиве, а потом указывается количество элементов в одномерных массивах.

Схема выделения памяти под двумерный массив **int A[20][5];**

- ▪ Элементами первого одномерного массива являются адреса, а второго – целые значения.



5. Указатели и массивы. Примеры

Пример 1. *Разработать программу считывания строк разной длины с использованием арифметики указателей*.

Программный код решения примера:

```
#include <stdio.h>

int main (void)
{
   int i, n;
   char *ptr[ ] = {"one", "two", "three", "four", "five",\
         "six", "seven", "eight", "nine", "ten"};

   n = sizeof(ptr)/sizeof(ptr[0]);

   printf("\n\t Strings of various length:\n");
   for (i = 0; i < n; ++i)
       printf("\n%12d) %s", i+1, ptr[i]);

   printf("\n\n Press any key: ");
   getch();
   return 0;
}
```

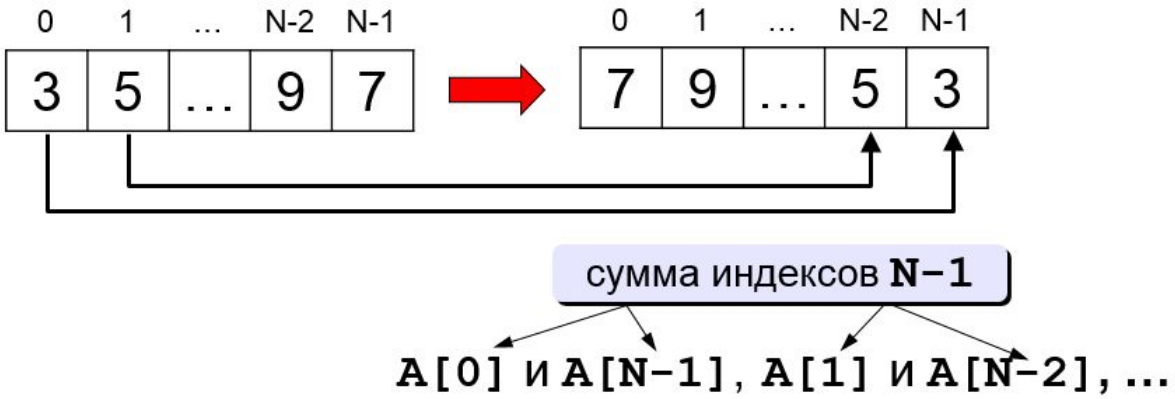
- В программе использован **одномерный массив указателей**.
- Функция **printf()** и спецификатор преобразования **%s** допускают использование в качестве параметра указатель на строку.
- При этом на дисплей выводится не значение указателя, а содержимое адресуемой им строки.
- **Обратный слеш** "****" служит для переноса содержимого операторной строки на новую строку (для удобства восприятия).
- Оператор **sizeof()** вычисляется во время компиляции программы.
- Во время компиляции **sizeof()** обычно превращается в **целую константу**, значение которой равно размеру типа или объекта, в *данном случае соответствует* **размеру массива указателей**.
- Следует обратить внимание на **инициализацию** *массива указателей*.
- Содержимое, заключенное в фигурные скобки, представляют собой **строки**, для каждой из которой служит указатель, входящий в **массив указателей**.
- Результат выполнения программы показан ниже:

```
        Strings of various length:

           1) one
           2) two
           3) three
           4) four
           5) five
           6) six
           7) seven
           8) eight
           9) nine
          10) ten

Press any key:
```

4. Примеры обработки массивов

1. Реверс массива

Задача: *переставить элементы массива в обратном порядке (выполнить инверсию).*



Алгоритм:
поменять местами **A[0]** и **A[N-1]**, **A[1]** и **A[N-2]**, …

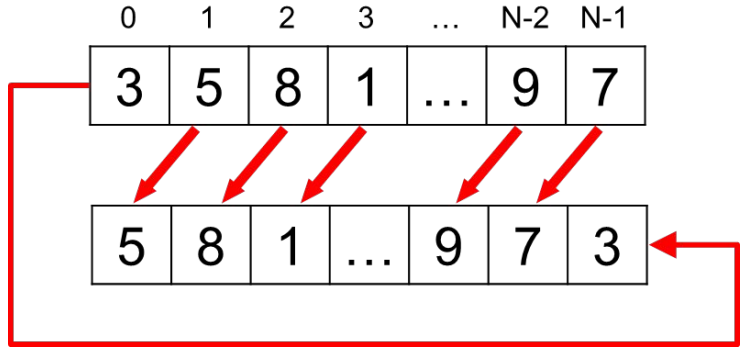
Фрагмент кода:

for (i = 0; i < N/2; i++) // поменять местами A[i] и A[N-1-i]

```
main()
{
  const int N = 10;
  int A[N], i, c;
  // заполнить массив
  // вывести исходный массив
  for ( i = 0; i < N/2; i++ )
{
    c = A[i];
    A[i] = A[N-1-i];
    A[N-1-i] = c;
    }
  // вывести полученный
массив
}
```

2. Циклический сдвиг

Задача: *сдвинуть элементы массива влево на 1 ячейку, первый элемент становится на место последнего.*



Алгоритм:
A[0]=A[1]; A[1]=A[2];… A[N-2]=A[N-1];

Фрагмент кода:

```
main()
{
  const int N = 10;
  int A[N], i, c;
  // заполнить массив
  // вывести исходный массив
  c = A[0];
  for ( i = 0; i < N-1; i ++)
        A[i] = A[i+1];
  A[N-1] = c;
  // вывести полученный массив
}
```

3. Сортировка массивов

На ПЗ и ЛР

4. Поиск в массиве

На ПЗ и ЛР

Примеры обработки массивов

Пример 1: все элементы массива увеличиваются на 1.

```
int  A[3] = {1, 2, 3};
```

Вариант 1

```
A[0] = A[0]+1;
A[1]+=1;
A[2]++;
```

Вариант 2

```
A[0]++;
A[1]++;
A[2]++;
```

Вариант 3

```
for ( int  i = 0;  i < 3;  i++ )
      A[i]++;
```

Пример 2:

```
double  m[100], a, b;
…
b = 3 * m[2];
a = m[50] / b;
m[99]++;
```

Пример 3: сложение двух массивов.

```
int  A[4] = { 2, 3, 4};
int  B[] = {1, -1, 5};
int C[4] = {0};
```

Вариант 1

```
C[0] = A[0]+B[0];
C[1] = A[1]+B[1];
C[2] = A[2]+B[2];
```

Вариант 2

```
for (int i=0; i<3; i++)
    C[i] = A[i]+B[i];
```

Использование массивов

- Используются для хранения различных последовательностей
- Обработка массивов
- Сортировка массивов
- Поиск в массиве
- и др.

Примеры обработки массивов

Рассмотрим задачу «инвертирования» массива символов и различные способы ее решения с применением указателей (заметим, что задача может быть легко решена и без указателей - с использованием индексации).

Предположим, что длина массива типа **char** равна **80**.

Первое решение задачи инвертирования массива:

```
char z[80], s;
char *d, *h;
/* d и h — указатели на символьные объекты */
for (d=z, h=&z[79]; d<h; d++, h--)
{   s = *d;
  *d = *h;
  *h = s;
}
```

- ✔ В заголовке цикла указателю **d** присваивается адрес первого (с нулевым индексом) элемента массива **z**.
- ✔ Здесь можно было бы применить и другую операцию, а именно **d=&z[0].**
- ✔ Указатель **h** получает значение адреса последнего элемента массива **z**.
- ✔ Далее работа с указателями в заголовке ведется как с обычными целочисленными переменными.
- ✔ Цикл выполняется до тех пор, пока **d<h**.
- ✔ После каждой итерации значение **d** увеличивается, значение **h** уменьшается на **1**.
- ✔ При первой итерации в теле цикла выполняется обмен значений **z[0]** и **z[79]**, так как **d** - адрес **z[0]**, **h** - адрес **z[79]**.
- ✔ При второй итерации значением **d** является адрес **z[1]**, для **h** - адрес **z[78]** и т. д.

Второе решение задачи инвертирования массива:

```
char z[80], s, *d, *h;
for (d=z, h=&z[79]; d<h;)
{
   s = *d;   *d++ = *h;   *h-- = s;
}
```

- ✔ Приращение указателя **d** и уменьшение указателя **h** перенесены в тело цикла.
- ✔ Напоминаем, что в выражениях ***d++** и ***h--** операции увеличения (**постфиксный инкремент**) и уменьшения (**постфиксный декремент**) на **1** имеют тот же приоритет, что и унарная адресная операция '*'.
- ✔ Поэтому изменяются на **1** не значения элементов массива, на которые указывают **d** и **h**, а сами **указатели**.
- ✔ Последовательность действий такая:
  - по значению указателя **d** (или **h**) обеспечивается доступ к элементу массива;
  - в этот элемент заносится значение из правой части оператора присваивания;
  - затем увеличивается (уменьшается) на **1** значение указателя **d** (или **h**).

Третье решение задачи инвертирования массива (используется цикл с предусловием):

```
char z[80], s, *d, *h;
d=z;
h=&z[79];
while (d < h)
{
   s=*d;   *d++ = *h;  *h-- = s;
}
```

Примеры обработки массивов

Продолжение (см. предыдущий слайд)

Рассмотрим задачу «инвертирования» массива символов и различные способы ее решения с применением указателей (заметим, что задача может быть легко решена и без указателей - с использованием индексации).

Предположим, что длина массива типа **char** равна **80**.

Четвертое решение задачи инвертирования массива (имитация индексированных переменных указателями со смещениями):

```
char z[80],s;
int i;
for (i=0; i<40; i++)
{
  s = *(z+i);
  *(z+i) = *(z+(79-i));
  *(z+(79-i)) = s;
}
```

Этот пример демонстрирует возможность использования вместо индексированного элемента **z[i]** выражения ***(z+i).**

В языке **Си**, как мы знаем, *имя массива без индексов есть адрес его первого элемента (с нулевым значением индекса).*

- ✔ Прибавив к имени массива целую величину, получаем адрес соответствующего элемента, таким образом, **&z[i]** и **z+i** – это две формы определения адреса одного и того же элемента массива, отстоящего на i позиций от его начала.
- ✔ Итак, в соответствии с синтаксисом языка операция индексирования **E1[E2]** определена таким образом, что она эквивалентна ***(E1+E2)**, где **E1** - имя массива, **E2** - целое.
- ✔ Для многомерного массива правила остаются теми же.
- ✔ Таким образом, **E[n][m][k]** эквивалентно ***(E[n][m]+k)** и, далее, ***(*(*(E+n)+m)+k)**.

- ❑ **Имя массива** не является переменной типа указатель, а есть ***константа-адрес начала массива***.
- ❑ Таким образом, к имени массива не применимы операции **'++'** (увеличения), **'--'** (уменьшения), имени массива нельзя присвоить значение, то есть имя массива не может использоваться в левой части оператора присваивания.
- ❑ В рассмотренных выше примерах указатели относились к символьным переменным, и поэтому их приращения были единичными.
- ❑ Однако это обеспечивалось только особенностями представления в памяти **символов** (***char***) – каждый символ занимает в памяти один байт, и поэтому адреса смежных элементов символьного массива отличаются на **1**.
- ❑ В случае массивов с другими элементами (**другого типа**) единичному изменению указателя, как уже отмечалось, соответствует большее изменение адреса.

Примеры обработки массивов

Задача: *ввести с клавиатуры массив из 5 элементов, умножить все элементы на 2 и вывести полученный массив на экран*.

```
#include <stdio.h>
#include <conio.h>
void main()
{
const int N = 5;
int A[N], i;
 // ввод элементов массива
printf("Введите 5 элементов массива:\n
for( i=0; i < N; i++ ) {
  printf ("A[%d] = ", i );
  scanf ("%d", & A[i] );
  }
 // обработка массива
printf("Введите 5 элементов массива:\n");
for( i=0; i < N; i++ ) {
  printf ("A[%d] = ", i );
  scanf ("%d", & A[i] );
  }
 // вывод результата
printf("Введите 5 элементов массива:\n");
for( i=0; i < N; i++ ) {
  printf ("A[%d] = ", i );
  scanf ("%d", & A[i] );
  }
getch();
}
```

```
A[0] = 5
A[1] = 12
A[2] = 34
A[3] = 56
A[4] = 13
```

Результат:

```
10  24  68 112  26
```

Самостоятельно дома:

1) Ввести с клавиатуры массив из 5 элементов, найти среднее арифметическое всех элементов массива.

```
Введите пять чисел:
4  15  3  10  14
среднее арифметическое 9.200
```

2) Ввести с клавиатуры массив из 5 элементов, найти минимальный из них.

```
Введите пять чисел:
4  15  3  10  14
минимальный элемент 3
```

Примеры обработки массивов. Заполнение случайными числами

```
#include <stdlib.h>   // случайные числа
```

RAND_MAX – максимальное случайное целое число (обычно **RAND_MAX = 32767**)

Случайное целое число в интервале [0,RAND_MAX]

```
x = rand(); // первое число
x = rand(); // уже другое число
```

Установить начальное значение последовательности:

```
srand ( 100 ); /* инициализирует генератор случайных чисел начальным числом*/
```

Целые числа в интервале [0,N-1]:

```
int random(int N)
{
  return rand()% N;
}
```

Примеры:

```
x = random ( 100 );  // интервал [0, 99]
x = random ( z );     // интервал [0, z-1]
```

Целые числа в интервале [a,b]:

```
x = random ( z ) + a;           // интервал [a, z-1+a]
x = random (b – a + 1) + a; // интервал [a, b]
```

NB: начальное число можно изменить с помощью функции **srand()**, которой в качестве параметра передается любое целое число. Вы можете сами (!) задавать инициализирующее значение, например: **srand(time(NULL));** /* из текущего времени получили целое число (параметр NULL); **#include <time.h>** */

Пример: Заполнение случайными числами

```
#include <stdio.h>
#include <stdlib.h>

/* функция выдает случайное число от 0 до N-1 */
int random(int N)
{ return rand() % N; }

main()
{
 const int N = 10;
 int A[N], i;
 printf("Исходный массив:\n");
 for (i = 0; i < N; i++ )
 {
   A[i] = random(100) + 50;
   printf("%4d", A[i]);
 }
...
}
```

Самостоятельно дома:

3) Заполнить случайными числами [100, 150]. Найти максимальный элемент и его номер

4) Заполнить массив из 10 элементов случайными числами в интервале [-10..10] и найти в нем максимальный и минимальный элементы и их номера.

ЛИТЕРАТУРА

1. Демидович Е. Основы алгоритмизации и программирования. Язык Си: учебное пособие – СПб.: БХВ – Петербург, 2006. – 440с.
2. Жешке Р. Толковый словарь стандарта языка Си. – СПб.: Питер, 1994. – 221с.
3. Керниган Б., Ритчи Д. Язык программирования Си: Пер. с англ. – 2-е изд., перераб. и доп. – М.: Финансы и статистика, 1992. – 272с.
4. Кочан С. Программирование на языке С, 3-е издание: Пер. с англ. – М.: ООО “И. Д. Вильямс”, 2007. – 496с.
5. Подбельский В., Фомин С. Программирование на языке Си: учебное пособие. 2-е доп. Изд. – М: Финансы и статистика, 2001. – 2001. – 600с.
6. Прата С. Язык программирования С. Лекции и упражнения, 5-е издание.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2006. – 960с.
7. Шилдт Г. Полный справочник по С. 4-е издание.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 704с.
8. Харбисон С., Стил Г. Язык программирования С.: Пер. с англ. – М: ООО Бином Пресс, 2004. – 528с.